Thuộc tính phương thức private và protected

Một trong những nguyên tắc quan trọng trong lập trình hướng đối tượng - phân chia rõ ràng giữa giao diện (khả năng truy cập) bên ngoài và trong.

Interal and external interface

Trong lập trình hướng đối tượng, thuộc tính và phương thức được chia thành 2 nhóm:

- Internal interface - phương thức và thuộc tính chỉ có thể được truy cập bên trong các phương thức trong class, không phải từ bên ngoài.
- External interface - phương thức và thuộc tính có thể truy cập được từ ngoài và trong class.

Trong Javascript, có 2 loại thuộc tính và phương thức:

- Public: có thể truy cập từ bất kỳ đâu. Nghĩa là external interface. Cho đến bây giờ thì chúng ta chỉ sử dụng thuộc tính public
- Private: có thể truy cập bên trong class. Nghĩa là internal interface

Trong nhiều ngôn ngữ khác thì còn tồn tại trường "protected": chỉ có thể truy cập bên trong class và những class kế thừa.

Trường Protected không được quy định trong Javascript ở cấp độ ngôn ngữ, những trong thực tế để cho tiện lợi thì chúng ta có thể giả lập để quy ước với nhau.

Protected

Điều này không được đảm bảo về cấp độ ngôn ngữ, nhưng giữa các lập trình viên thống nhất với nhau. Các thuộc tính protected thì thường bắt đầu với tiền tố là `_`

Ví dụ ở đây ta có thuộc tính protected là `_waterAmount`

```
class CoffeeMachine {
  _waterAmount = 0

  set waterAmount(value) {
    if (value < 0) {
      value = 0
    }
    this._waterAmount = value
  }

  get waterAmount() {
    return this._waterAmount
  }

  constructor(power) {
    this._power = power
  }
```

```
}

// create the coffee machine
let coffeeMachine = new CoffeeMachine(100)

// add water
coffeeMachine.waterAmount = -10 // Error: Negative water
```

Chúng ta truy cập thông qua getter/setter. Đồng thời có thể thêm điều kiện khi setter

Readonly

Chúng ta tạo thuộc tính `power` là read-only tức là chỉ có thể get chứ không thể set. Lưu ý: điều này cũng không được đảm bảo về mặt ngôn ngữ

```
class CoffeeMachine {
  // ...

  constructor(power) {
    this._power = power
  }

  get power() {
    return this._power
  }
}

// create the coffee machine
let coffeeMachine = new CoffeeMachine(100)

alert(`Power is: ${coffeeMachine.power}W`) // Power is: 100W

coffeeMachine.power = 25 // Error (no setter)
```

Getter/setter functions

Hầu hết mọi lần thì chúng ta thường thích dùng `get.../set...` hơn, như thế này

```
class CoffeeMachine {
  _waterAmount = 0

  setWaterAmount(value) {
    if (value < 0) value = 0
    this._waterAmount = value
  }

  getWaterAmount() {
    return this._waterAmount
  }
```

```
}

new CoffeeMachine().setWaterAmount(100)
```

Nó trông dài hơn những function thì đa năng hơn, bạn có thể truyền nhiều tham số. Cú pháp get/set thì ngắn hơn, như nó có quy tắc bị giới hạn bởi tham số truyền vào.

Protected fields được kế thừa

Nếu chúng ta kế thừa `class MegaMachine extends CoffeeMachine`, thì không có gì có thể ngăn cản chúng ta truy cập `this._waterAmount` hoặc `this._power` từ các phương thức của class mới.

Vì thế các trường protected thì có thể được thừa kế. Không như các trường private dưới đây.

Private

Tính năng này được thêm gần đây. Không được hổ trở bởi nhiều JS engine, vì thế cần polyfill

Cuối cùng thì chúng ta cũng có một thuộc tính "riêng tư" được hổ trợ bởi chính ngôn ngữ JS.

Private thì bắt đầu bằng `#`. Nó chỉ có thể truy cập bên trong class.

Ví dụ thuộc tính private `#waterLimit` và phương thức private `#checkWater`

```
class CoffeeMachine {
  #waterLimit = 200

  #fixWaterAmount(value) {
    if (value < 0) return 0
    if (value > this.#waterLimit) return this.#waterLimit
  }

  setWaterAmount(value) {
    this.#waterLimit = this.#fixWaterAmount(value)
  }
}

let coffeeMachine = new CoffeeMachine()

// can't access privates from outside of the class
coffeeMachine.#fixWaterAmount(123) // Error
coffeeMachine.#waterLimit = 1000 // Error
```

Các trường private thì không bị xung đột với các trường public. Chúng ta có thể khai báo private `#waterAmount` và public `waterAmount` cùng 1 lúc.

```
class CoffeeMachine {
  #waterAmount = 0
```

```
  get waterAmount() {
    return this.#waterAmount
  }

  set waterAmount(value) {
    if (value < 0) value = 0
    this.#waterAmount = value
  }
}

let machine = new CoffeeMachine()

machine.waterAmount = 100
alert(machine.#waterAmount) // Error
```

Không như các trường protected, private fields được đảm bảo về mặt ngôn ngữ. Nếu chúng ta kế thừa từ `CoffeeMachine`, thì chúng ta sẽ không truy cập được vào `#waterAmount`

```
class MegaCoffeeMachine extends CoffeeMachine {
  method() {
    alert(this.#waterAmount) // Error: can only access from CoffeeMachine
  }
}
```

Lưu ý: bình thường chúng ta có thể truy cập vào các thuộc field dạng `this[name]`:

```
class User {
  ...
  sayHi() {
    let fieldName = "name";
    alert(`Hello, ${this[fieldName]}`);
  }
}
```

Với các trường private thì không thể: `this['#name']` sẽ không hoạt động.